Gymnázium, Praha 6, Arabská 14

Programování

MATURITNÍ PROJEKT

WaitR

2024/2025 **Michael Vakula 4.E**

Gymnázium, Praha 6, Arabská 14

Programování

MATURITNÍ PROJEKT

WaitR

Předmět: Programování

Téma: Waitr

Autor: Michael Vakula

Třída: 4.E

Školní rok: 2024/25

Vedoucí práce: Mgr. Jan Lána

Třídní učitel: Mgr. Blanka Hniličková

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium, Praha 6, Arabská14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V …..... dne …..... Michael Vakula .....

Anotace

Cílem mé maturitní práce bylo vytvořit aplikaci pro efektivní práci číšníka umožňující snadnou správu restaurace v reálném čase. Uživatelé mohou zakládat podniky, vytvářet 2D modely s rozložením stolů a orientačními prvky, evidovat zákazníky a spravovat jejich objednávky. Menu je přizpůsobitelné a stav objednávek (např. doručení jídla nebo platba) je okamžitě synchronizován mezi všemi zařízeními. Aplikace navíc nabízí analytické funkce, jako je sledování oblíbenosti jídel nebo hodnocení výkonu číšníků.

Abstract

The goal of my graduation project was to create an application for efficient waiter work, enabling real-time restaurant management. Users can establish businesses, create 2D models with table layouts and reference points, track customers, and manage their orders. The menu is customizable, and the status of orders (e.g., meal delivery or payment) is instantly synchronized across all devices. Additionally, the application provides analytical features, such as monitoring dish popularity or evaluating waiter performance.

Zadání

Aplikace pro restaurace umožní uživatelům zakládat organizace (podniky) a přidávat další účastníky. Každý podnik si může vytvořit 2D model restaurace pomocí přidání stolů a orientačních prvků. U jednotlivých stolů bude možné zaznamenávat zákazníky, jejich objednávky (menu bude spravovatelné majitelem) a sledovat stavy, jako zda zákazníci již dostali jídlo nebo zaplatili. Aplikace bude fungovat v reálném čase s online synchronizací mezi zařízeními (změny provedené jedním číšníkem budou okamžitě viditelné pro ostatní). Součástí bude také analytická funkce, která umožní sledovat frekvenci objednávek jednotlivých jídel nebo vyhodnocovat efektivitu číšníků.

Obsah

1. Úvod....8
2. Použité technologie....9
3. Struktura projektu....10
4. Popis hlavních scén....11
    4.1. Company menu....11
    4.2. Model view....12
    4.3. Food menu....13
    4.4. Analytics....14
    4.5. Notifikace....15
    4.6. Company settings....15
5. Datové struktury....17
    5.1. Databáze....17
    5.2. Model podniku....17
    5.3. Jídelní menu....18
    5.4. Struktura pro notifikace....19
    5.5. Struktura pro nastavení....19
6. Popis funkcí aplikace a řešení problémů....20
    6.1. Registrace a přihlášení....20
    6.2. Načítání společností....20
    6.3. Pozvánkový systém....21
    6.4. Obsluha fragmentů....22
    6.5. Obsluha NavigationMenu....24

6.6. Načítání dynamických dat.....25
6.7. Správa notifikací.....26
6.8. Editace.....27
6.8.1. Pozice a rozměry.....27
6.9. Vykreslení grafů.....28
7. Závěr.....29
Zdroje
Seznam obrázků

1. Úvod

Nápad na tuto ročníkovou práci vznikl na základě mé vlastní zkušenosti s prací číšníka v restauraci, kde jsem často zápasil s nepřehledným číslováním stolů a chaotickým systémem objednávek. Během služeb jsem si uvědomil, jak moc by podobným situacím pomohla jednoduchá mobilní aplikace, která by usnadnila evidenci stolů, správu objednávek a celkově zlepšila efektivitu práce.

Vím, že mnoho moderních podniků využívá profesionální restaurační systémy, ale moje aplikace si neklade za cíl je nahradit. Místo toho může sloužit jako užitečný pomocník pro brigádníky v menších podnicích, které takový systém nemají, nebo jako osobní organizér pro číšníky, kteří si chtějí udržovat přehled o svých stolech přímo v mobilu.

Cílem této práce je vytvořit intuitivní nástroj, který zjednoduší každodenní provoz v restauraci, sníží chybovost a ušetří čas jak číšníkům, tak celému týmu.

2. Použité technologie

Jako vývojové prostředí bylo zvoleno Android Studio, což je oficiální IDE pro tvorbu aplikací na platformě Android. Poskytuje nástroje pro návrh uživatelského rozhraní, ladění, testování a optimalizaci výkonu. Verze SDK Ladybug přináší vylepšení v oblasti stability a vývojářských nástrojů, což usnadňuje vývoj moderních Android aplikací [1].

Jako hlavní programovací jazyk byl zvolen Kotlin, což je moderní programovací jazyk, který se stal preferovanou volbou pro vývoj Android aplikací. Google jej oficiálně podporuje a nabízí lepší čitelnost, stručnost a bezpečnost oproti Javě. Jeho popularita mezi vývojáři roste, což z něj dělá ideální volbu pro tento projekt [2].

Pro databázi jsem zvolil cloudovou platformu Firebase od Googlu, která poskytuje backendové služby pro mobilní a webové aplikace. V této práci byly využity zejména Firebase Authentication pro správu uživatelů a Firebase Realtime Database, což je NoSQL databáze umožňující synchronizaci dat v reálném čase mezi zařízeními. Díky tomu je možné okamžitě aktualizovat objednávky a stavy stolů bez nutnosti ručního obnovování [3].

MPAndroidChart je open-source knihovna pro vytváření interaktivních grafů v Android aplikacích. Je dobře dokumentovaná a široce používaná, což z ní dělá vhodné řešení pro analytickou část aplikace [4].

3. Struktura projektu

V této kapitole se zaměřím na strukturu souborů v mém projektu. Při vytváření nového projektu v Android studio se kromě části app, kde zejména probíhá vývoj aplikace, vytvoří také mnoho souborů sloužící pro konfiguraci a figuruje zde hlavně gradle, což je moderní nástroj pro automatizaci sestavování projektů, který v Android vývoji slouží ke kompilaci kódu, správě externích knihoven, optimalizaci aplikace pro různé verze a zjednodušení celého procesu vývoje tím, že automaticky řeší závislosti a urychluje nasazování aplikace [5]. Zejména důležitý soubor spojený s gradlem je soubor build.gradle.kts, který slouží pro konfiguraci a přidávání knihoven do projektu.

Ve složce app nalezneme hlavní složky a soubory, které definují naši aplikaci, a to java, res a AndroidManifest.xml.

Soubor AndroidManifest je klíčovou součástí aplikace, protože definuje její strukturu, metadata, komponenty a požadavky. V tomto souboru můžeme nalézt uzly pro všechny aktivity, služby a poskytovatele obsahu, které aplikaci tvoří [6].

Ve složce java najdeme všechny třídy a datové struktury, které tvoří logiku aplikace a zpracovávají všechny vstupy a výstupy. Hlavní třídy v programu dědí třídu Aktivity a fragment, kde aktivity je základní komponentou, která spravuje uživatelské rozhraní a interakce aplikace a funguje na bázi volání specifických callback metod, které odpovídají jednotlivým fázím životního cyklu aktivity jako onStart(), onPause() a onDestroy(). Fragment představuje modulární část uživatelského rozhraní v rámci aktivity a má svůj vlastní životní cyklus, přijímá své vlastní vstupní události a můžeme jednotlivé fragmenty odpojovat a připojovat k aktivitě [7].

Složka res slouží k definování vzhledu uživatelského rozhraní, kde můžeme najít podsložky pro jednotlivé layouty, což jsou něco jako scény, a drawable elementy jako různé ikonky atd.

4. Popis hlavních scén

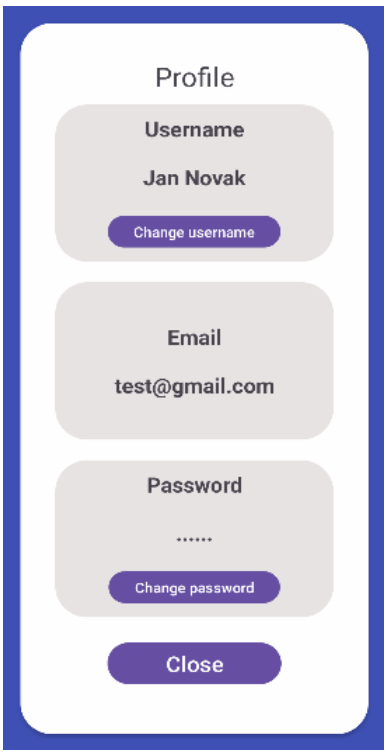
V této kapitole budu popisovat většinu scén a budu se snažit přiblížit postup a princip, jak správně používat jednotlivé části aplikace

4.1. Company menu

Po úspěšném přihlášení je uživatel přenesen na scénu company menu, kde lze vidět všechny podniky, ve kterých je uživatel členem. Podniky jsou rozděleny podle barvy na ty, kde je uživatel majitelem (modré) a ty kde je buď zaměstnancem nebo manažerem (zelené). Majitel podniku má na rozdíl od ostatních možnost podnik smazat, jinak se jednotlivé funkce nijak nemění. Dále na scéně vidíme dvě tlačítka, a to pro vytvoření nového podniku a pro připojení se k podniku přes prostřednictví pozvánky.

Na této scéně také nalezneme postranní menu, kde je možné se odhlásit, nebo změnit své profilové údaje mezi které patří uživatelské jméno a heslo.

Obr. 1: Company menu scéna



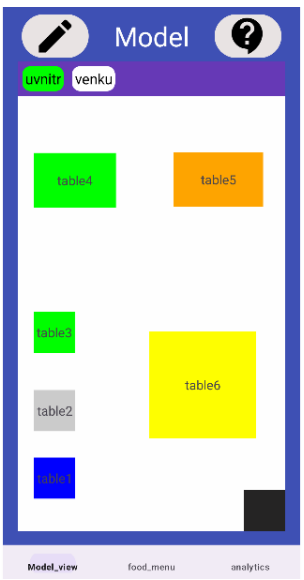
Obr. 2: profile settings

4.2. Model view

Tato scéna je snad ta nejdůležitější v celé aplikaci, jelikož zde se vytváří a edituje model rozložení stolů v podniku a probíhají tu všechny operace, které se týkají obsluhy stolů. Je zde vidět horní lišta kde jsou zobrazeny jednotlivé části modelu, na které lze kliknout a podle toho se vykreslí stoly. Nad lištou jdou vidět dvě tlačítka, kde jedno slouží k editaci modelu a druhé k zobrazení nápovědy k použití. Samotné stoly lze vidět v pěti různých barvách:

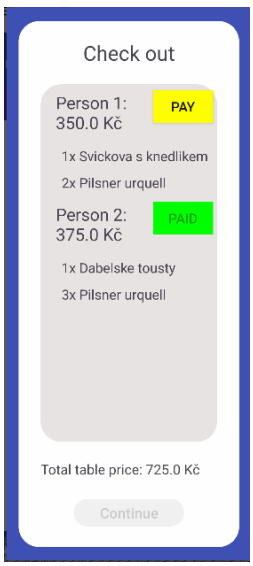
- Šedá – stůl je prázdný
- Žlutá – stůl je obsazen novými zákazníky
- Oranžová – zákazníci si objednali a čekají
- Zelená – zákazníci jedí
- Modrá – zákazní zaplatili a odešli a stůl je nutné uklidit

Jeden celý cyklus obsluhy stolu probíhá tak, že nejdřív se u stolu vybere počet zákazníků od 1 do 10 a pro každého jednoho zákazníka se vytvoří jeho seznam objednávek, kam se můžou přidávat a odebírat položky z jídelního menu, číšník si u položek může i poznamenávat, jestli jsou už přineseny ke stolu, nebo ne. Když už všichni dojedli je možné jít na checkout, kde je přehledně znázorněno kdo co kolik měl a kolik má zaplatit. Po zaplacení se stůl špinavý a říká číšníkovi, aby stůl uklidil pro konečné uvolnění.



Obr. 3: model

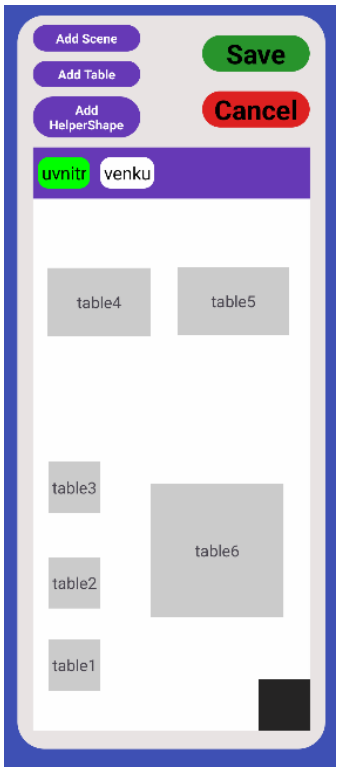
Obr. 4: list objednavek u stolu



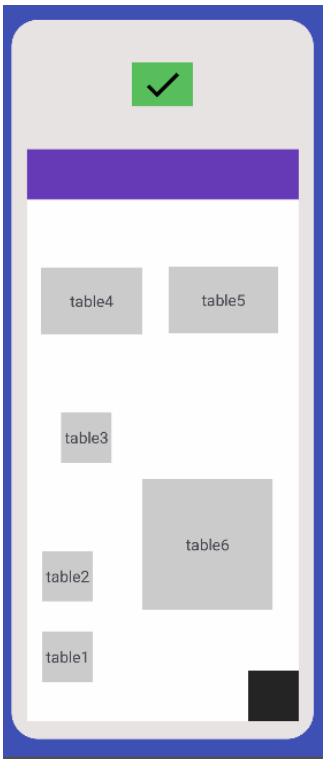
Obr. 5: proces placení

Kromě stolů model obsahuje také takzvané pomocné útvary, jejich myšlenka využití slouží pro lepší vyobrazení prostotu restaurace a aby se uživatel dokázal lépe orientovat v prostoru a mohl si vyznačit kde jsou například dveře, sloupy, zdi atd.

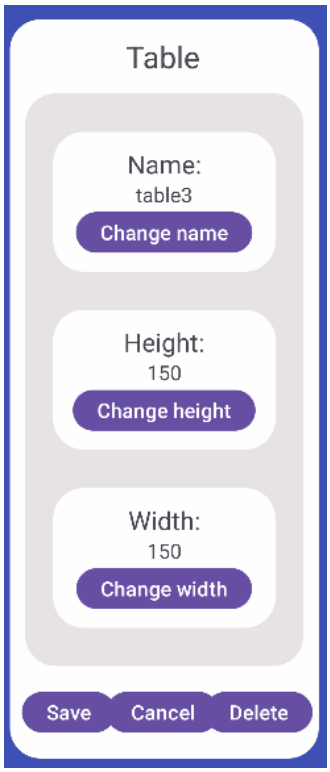
Při editaci modelu můžeme vidět horní lištu, kde najdeme tlačítka pro uložení a zrušení změn, vytvoření nového plátna a pro přidání nových stolů a pomocných útvarů. Jak stoly, tak pomocné útvary můžeme editovat dvěma způsoby. První způsob je, že po kliknutí na stůl či pomocný útvar, se nám změní mód na přesun a můžeme libovolně přesouvat pozici objektu. Druhý způsob je při takzvaném dvojitém kliknutí a umožňuje nám měnit výšku a šířku objektu, objekt smazat, nebo u stolu změnit jeho název.



Obr. 6: editace modelu



Obr. 7: editace pozice stolu

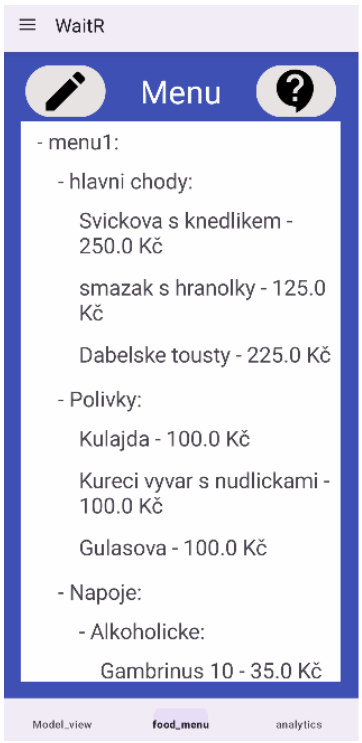


Obr. 8: editace parametrů stolu

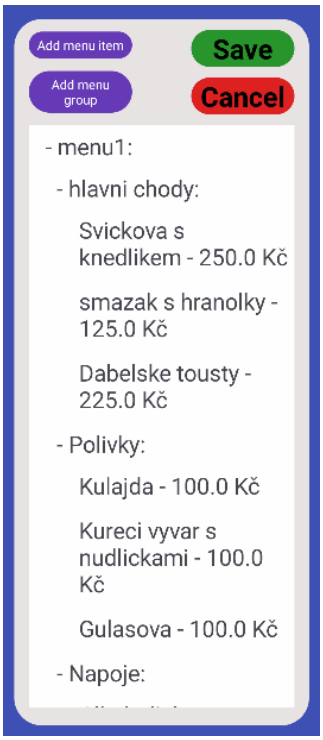
4.3. Food menu

Tato scéna slouží k vytvoření vlastního jídelního menu a umožňuje vytvářen skupiny pro přehlednější rozložení položek a lepší orientaci. U jednotlivých položkách v menu je možné nastavovat název, popis a cenu.

Při editaci jídelního meny uživatel vidí podobnou scénu jako u modelu, ale s rozdílem, že přidává pouze skupiny a položky. Pro vybírání, do jaké skupiny položku přidat stačí pouze kliknout na danou skupinu, možné je do skupin přidávat další podskupiny pro další členění. Také tu je možnost dvojkliku, se kterým lze editovat či mazat jak skupiny, tak položky.



Obr. 9: ukázka jídelního meny

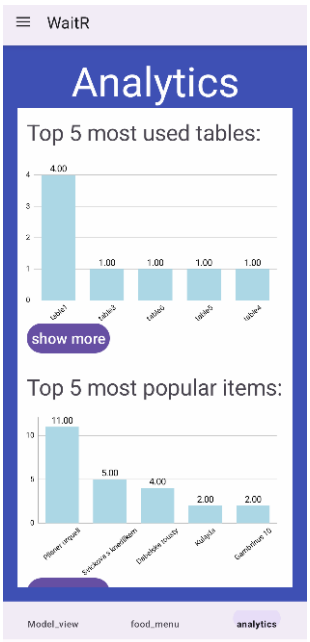


Obr. 10: editace jídelního meny

4.4. Analytics

V této scéně se vykreslují grafy pro zobrazení sbíraných dat. Mezi informace, které aplikace sbírá, patří následující data. Pro každý stůl se zaznamenává kolikrát již byl navštíven a umožňuje uživateli sledovat popularitu u jednotlivých stolů, to samé platí pro jídlo a sbírají se data o popularitě položek z jídelního meny. U každého uživatele se zaznamenávají dvě informace, a to kolik stolů obsloužil a takzvanou aktivitu. Aktivita je určena počtem potvrzených notifikacích, které uživateli napovídají že by měl zkontrolovat stůl. Princip fungování notifikací vysvětlím podrobněji v následující kapitole.

Data jsou zobrazeny ve formě sloupcového grafu a ukazuje se vždy prvních pět s největším hodnocením. Lze si zobrazit data i u ostatních s menším hodnocením.



Obr. 11: grafy

Obr. 12: zobrazení více dat

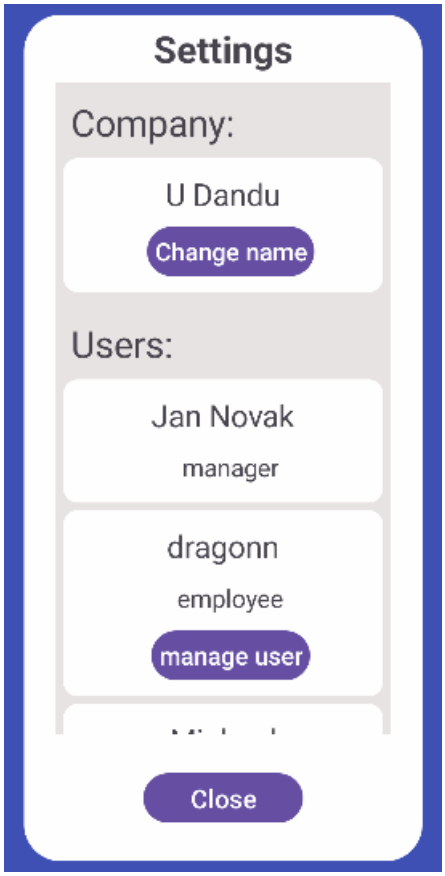
4.5. Notifikace

Notifikace v mé aplikaci slouží k upozornění uživatele, aby zkontroloval stůl. Jsou tři typy notifikace pro různá stádia stolu, a to, když stůl je obsazen, ale nemají objednáno a notifikace říká, aby se uživatel šel podívat, jestli jsi už vybrali, když zákazníci jedí a notifikace říká, jestli náhodou nechtějí něco dalšího, a když je stůl nutné uklidit. Za jak dlouho budou notifikace chodit je možné nastavit zvlášť, a to od rozmezí od pěti minut až po 45. Všechny notifikace se zobrazují v sekci notifikace v postranním menu aplikace s možností je potvrdit.

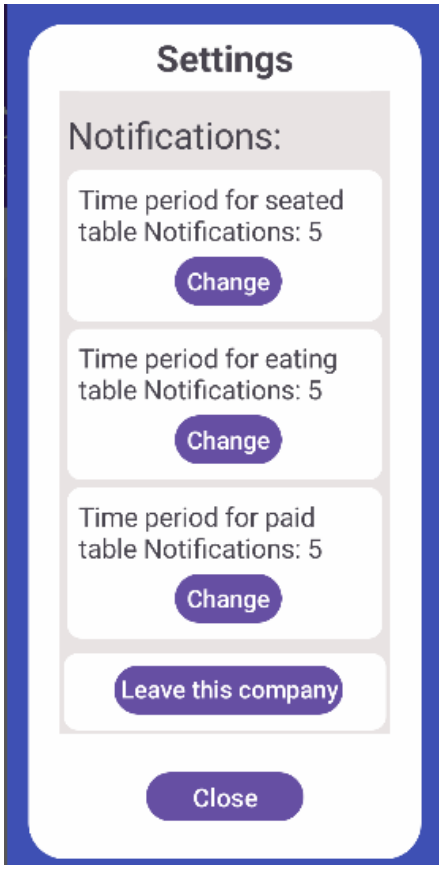
4.6. Company settings

Na této scéně se spravuje nastavení podniku. Lze zde měnit jméno podniku, nastavovat pozice uživatelům a vyhazovat je, a nastavovat velikost intervalu se kterým se posílají jednotlivé typy notifikací, a také možnost opustit podnik. Kdo může měnit nastavení podniku do určité míry je určeno pomocí pozic v podniku.

Jsou zde tři pozice a to vlastník, ten, co podnik založil a může dělat vše, dále je tu manažer, který má všechny pravomoci jako vlastník, ale nemůže spravovat účty vlastníka a ostatních manažerů, a pak zaměstnanec, který nemá přístup k nastavení podniku a může podnik pouze opustit, nemůže také editovat jak model, tak jídelní menu a nemůže pozvat nové uživatele.



Obr. 13: settings ukázka 1



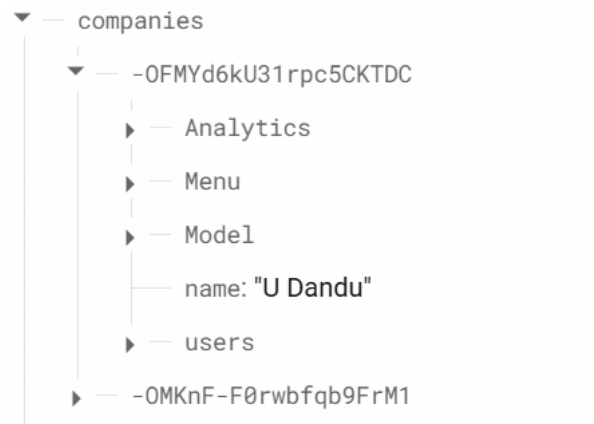
Obr. 14: settings ukázka 2

5. Datové struktury

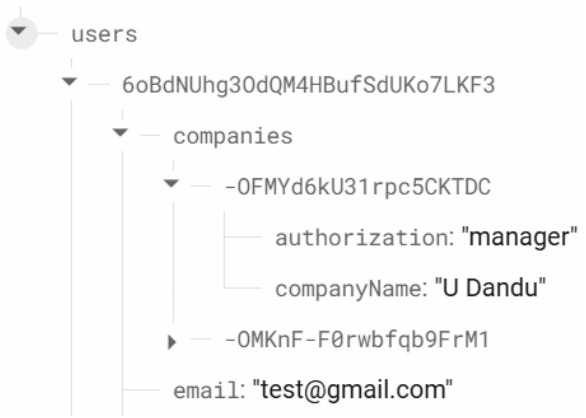
V této kapitole se budu věnovat popisu datových struktur v mém projektu.

5.1. Databáze

Jak už bylo jednou zmíněno, používám cloudovou nosql databázi realtime database od firebase, kde se data ukládají hierarchicky v JSON formátu a k určitým datům se chodí pomocí cesty, kde se zadávají jména takzvaných uzlů. Pro čtení se používá na chtěný uzel metoda getValue(), pro zápis nejčastěji updateChildren() a pro mazání deleteChildren(). V mé databázové struktuře jsou dva hlavní uzly, a to "users" pro uživatele a "companies" pro všechny společnosti. K velice používaným metodám pro práci s daty bylo využití serializace objektů do databáze, kde pokud má objekt metodu pro mapování jeho parametrů, je pak velmi jednoduché ukládat složité datové struktury do databáze.



Obr. 15: uzel "companies"



Obr. 16: uzel "users"

V uzlu "companies" mají všechny poduzly definovanou datovou strukturu, kterou si rozebereme v následujících podkapitolách. Co se týče uzlu "Analytics", ten obsahuje všechny už zmíněné typy dat pro vykreslování grafů. Poduzel "users" má kromě parametrů username, email a authorization, také parametr status, který říká, jestli je uživatel zrovna online nebo ne.

5.2. Model podniku

V této podkapitole se budu věnovat všem třídám, které tvoří model podniku. Jako první objekt, který nese všechny ostatní je objekty je tu třída Model.

Třída Model má pouze dva parametry, a to parametr "locked”, který slouží, aby když někdo z uživatelů editoval model, pro ostatní by editace byla pro určitou dobu nemožná. Zabraňuje to tak nesouvislému přepisu dat. Druhý parametr je list datového typu ModelScene.

ModelScene reprezentuje jednotlivé scény v modelu a obsahuje tak tři parametry, a to “id” pro unikátní String, list datového typu Table a list datového typu HelperShape.

Datové struktury Table a HelperShape jsou si velmi podobné, a tak projdu nejdříve parametry, které nalezneme v obou dvou objektech. Oba dva objekty mají parametr “id”, “height” a “width”, což určuje rozměry tělesa, a “xPosition”, “yPosition”, kam se ukládá procento výšky a šířky plátna pro stanovení pozice tělesa. Objekt table má následně navíc parametry “name”, pro uložení jména, “state”, pro aktuální stav stolu, “locked”, aby opět nemohli dva uživatelé najednou obsluhovat jeden stůl, “numberOfPeople”, pro určeni, kolik zákazníků je u stolu, “totalTablePrice”, pro ukládání celkové ceny všech objednávek pro stůl, a na konec list datového typu Customer.

Objekt Customer, jak z náznu napovídá reprezentuje zákazníka a obsahuje parametry “id”, “name” a parametr typu Order.

Order obsahuje parametr “totalPrice”, pro uložení celkové ceny objednávky, booleanovou hodnotu paid, pro uložení, zda byla objednávka zaplacena, nebo ne, a list datového typu MenuItem, tento objekt si podrobněji popíšeme v následující podkapitole.

5.3. Jídelní menu

Objekt MenuGroup je kmenová třída, která nese všechny ostatní objekty pro reprezentaci jídelního menu. Obsahuje opět parametr “id”, “locked”, “name” a list, který obsahuje objekty stejného typu MenuGroup, dá se tak říct, že je to takový strom s mnoha uzly a hodí se pro něj například rekurzivní průchod. Poslední parametr je list typu MenuItem.

MenuItem obsahuje parametry “id”, “name”, “price”, pro uložení ceny, “description”, rpto uložení popisu položky a booleanová hodnota “served”, pro uložení, zda byla položka přinesena na stůl.

5.4. Struktura pro notifikace

Tato struktura slouží pro reprezentaci notifikací, které se ukládají do databáze a obsahují různá data, která hrají roli v následném poslání skutečné notifikace v android aplikaci. Mezi její parametry patří “id” jako unikátní klíč, “tableId”, což obsahuje id stolu, pro který je notifikace určena, “tableName” ukládá jméno stolu, “type”, což určuje, o jaký typ notifikace se jedná a podle toho se určuje zpráva při poslání skutečné notifikace, “timeToSend” pro uložení času poslání notifikace v sekundách, a “send”, což je booleanová hodnotá, říkající, zda byla notifikace poslána nebo ne.

5.5. Struktura pro nastavení

Do této datové struktury se ukládají data, podle kterých se vykreslí scéna pro nastavení společnosti. Obsahuje jméno společnosti, intovou hodnotu period, kdy se má poslat znovu notifikace u všech tří typů, a list všech uživatelů podle struktury User, která obsahuje id, jméno a pozici v podniku.

Všechny zmiňované objekty mají většinou vždy metodu toMap(), pro zápis do databáze, a metodu deepCopy(), pro kopii dat. Zde je ukázka pro objekt MenuItem

```
//mapování parametrů pro zápis do databáze
fun toMap(): Map<String, Any?> {
    return mapOf(
        "id" to id,
        "name" to name,
        "price" to price,
        "description" to description,
        "served" to served

    )
}
```

Obr. 17: ukázka metody toMap()

```
//Kopie dat
    fun deepCopy(): MenuItem {
        return MenuItem(
            id = this.id,
            name = this.name,
            price = this.price,
            description = this.description,
            served = this.served
        )
    }
```

Obr. 18: ukázka metody deepCopy()

6. Popis funkcí aplikace a řešení problémů

V této kapitole budu popisovat klíčové metody pro fungování většiny podstatných funkcí. Budu se snažit vysvětlit důvody pro zvolený návrh a nastínit problematiku, kterou metody řeší.

6.1. Registrace a přihlášení

V aplikaci se nachází forma autentikace, což je proces ověření identity uživatele. V mé aplikace nalezneme klasickou autentikaci přes email a heslo. Uživatel při registraci zadává své údaje do view elementu s názvem TextInputEditText, který umožňuje získávat a zpracovávat input od uživatele. Tento prvek je velice užitečný a využívá se na mnoha místech v aplikaci. Získané údaje se dále zkontrolují, zda např jméno není moc dlouhé, nebo email má správnou formu atd. Pro další zpracování a uložení údajů byli použity knihovní metody od firebase authentication. Kromě firebase auth se uživatel ukládá i do realtime databáze kam se zapisuje po uspěšném vytvoření uživatel ve firebase auth [8].

Když se uživatel úspěšně přihláší, v aplikace se přepne aktivita na Company menu, to je v kódu dosaženo pomocí mechanismu Intent, který systému dokáže poslat zprávu pro start nové aktivity a ukončení té předešlé. Intent dokáže předávat mezi aktivitami různá data, což je velmi užitečné a celkově intent je velice dost využíván v aplikaci [9].

6.2. Načítání podniků

Když potřebujeme, aby se nějaké metody spustili hned při startu aktivity, lze toho dosáhnou přes použití override metody onCreate(), která se automaticky spustí právě při startu aktivity. V aplikaci se v této metodě často nacházejí metody pro vykreslení dynamických UI prvků, jak je to například u seznamu podniků. Stejný princip funguje i u fragmentů, ale u nich je lepší využívat metodu onViewCreated(), jelikož je to bezpečnější a všechny UI prvky do kterých vkreslujeme už jsou načteny.

Při kliknutí na tlačítko pro vstup do společnosti se přepíná aktivita pomocí zmíněného intent a v tomto případě se pomocí metody putExtra() do nové aktivity pošle id podniku. Tento krok je velmi důležitý, jelikož podle tohoto id se vykreslují všechny UI prvky a načítají se odpovídající data.

6.3. Pozvánkový systém

Vlastník vytvořeného podniku může posílat pozvánky ostatním uživatelům, aby se do jeho podniku připojili. V této podkapitole popíšu, jak tento proces probíhá a vysvětlím klíčové metody.

Při kliknutí na políčko invite se uživateli rozklikne takzvaný Dialog, což je mechanismus v andoridu, který slouží jako pop up okno. Při jeho použití se stav aktivity, ve které byl dialog zavolán přepne na onPause(), a je tak pozastavená. Do dialogu si pomocí metody inflate() dokážeme takzvaně nafouknout libobolný layout a umožnuje tak pohodlný přístup k jeho view prvkům. Referenci na prvky se získává jednoduše pomocí metody findViewById(), kde se zadá id view elementu, které má určené ve svém xml layout souboru. Dialog má dvě hlavní metody, a to metodu show(), kterou vždy volá metoda pro otevření dialogu, a metodu dismiss(), pro uzavření dialogu [10].

```
private fun showInviteUserPopUp(){
    // Vytvoření dialogu
    val dialog = Dialog( context: this)
    dialog.setContentView(R.layout.company_manager_invite_popup)

    // Nastavení velikosti dialogu
    dialog.window?.setLayout(
        (resources.displayMetrics.widthPixels * 0.95).toInt(),
        (resources.displayMetrics.heightPixels * 0.85).toInt()
    )
    // Reference na prvky v popup layoutu
    val inviteButton = dialog.findViewById<Button>(R.id.invite_user_button)
    val userToInvite = dialog.findViewById<TextInputEditText>(R.id.user_to_invite)
    val spinner: Spinner = dialog.findViewById(R.id.positionSpinner)
    val closeButton = dialog.findViewById<Button>(R.id.close_invite_users_button)
```

Obr. 19: ukázka metody pro vytvoření dialogu a nastavení referencí na jeho prvky

Dialog se v aplikaci využívá velmi často a je tak jedním z nejdůležitějších mechanismů v aplikaci.

V dialogu pro poslání pozvánky můžeme vidět TextInputEditView pro zadání emailu uživatele, kterého chceme pozvat, Spinner, což je view prvek, který slouží jako rozevírací menu, obsahuje dvě položky, “employee” a “manager”, které nám říkají, na jakou pozici chceme uživatele do podniku pozvat. Pro správné fungování Spinneru se vytváří v androidu takzvný Adapter, který definuje, jak budou položky v rozbaleném stavu vypadat a stará se o UI design Spinneru [11]. Poslední view prvek je tlačítko pro poslání samotné pozvánky. Při kliknutí na tlačítko se spustí mnoho akcí, které si podrobně popíšeme. Nejdříve se pomocí emailu získá uid uživatele z firebase auth, to slouží jako id a podle něho se zapisují uživatelé v realtime databázi. Po získání uid se projede seznam uživatelů na uzlu “companies” a zjistí se, jestli už uživatel není členem. Potom se jde na uzel “users”, kde se podívá, jestli uživatel na poduzlu “invites” nemá už pozvánku od dané společnosti, v databázi má pozvánka formu Stringu odpovídajícímu id podniku s parametrem “position”, tudíž se pouze kontroluje, zda uzel “invites” má některý poduzel s takovou hodnotou, a pokud ne, tak id do uzlu přidá jako nový poduzel.

Uživatel může své pozvánky vidět, když klikne na tlačítko “join company” v aktivitě Comapany menu. Při kliknutí na tlačítko se rozklikne dialog a spustí se metoda loadInvites(), která udělá dotaz na databázi na uzel “invites” a dostane data ve formě dataSnapshot, což je jako celý podstrom daného uzlu.

Pro každý poduzel následně vytvoří UI objekty, které se dají přidat do layoutu. Takzvané view elementy lze vytvářet dynamicky a chovají se jako každý jiný objekt. Pro grafické znázornění pozvánky se vytvoří Textview, což je prvek, který obsahuje text a říká uživateli od jakého podniku mu přišla pozvánka, a dvě tlačítka (Button), jedno pro odmítnutí pozvánky, které jednoduše smaže danou pozvánku z uzlu a UI prvky smaže, a pro přijmutí, které přidá prvně podnik k seznamu podniku u uživatele, a následně samotného uživatele do seznamu uživatelů u podniku, nakonec smaže pozvánku a přidá nový prvek do seznamu podniků jako vstup.

6.4. Obsluha fragmentů

V aktivitě Company manager figurují tři fragmenty, které se dynamicky přepínají. Pro jejich správné fungování se ale musí nastavit při startu aktivity. V této podkapitole popíšu jejich správu a manipulaci s nimi.

Hned při startu aktivity se vytvoří instance všech třech fragmentů podle jejich tříd a přidělí se jim id společnosti jako parametr, pro načtení odpovídajících dat. Po vytvoření se zobrazí pouze první fragment pro model podniku, ostatní dva jsou skryty

```
// Vytvoření fragmentů a předání CompanyID
if (savedInstanceState == null) {
    CompanyID.let { id ->
        if (id.isNotEmpty()) {
            modelView = Model_view.newInstance(id)
            foodMenu = Food_menu.newInstance(id)
            analytics = Analytics.newInstance(id)

            // Přidejte všechny fragmenty, ale zobrazte pouze výchozí
            supportFragmentManager.beginTransaction()
                .add(R.id.companyframelayout, modelView,  tag: "ModelView")
                .add(R.id.companyframelayout, foodMenu,  tag: "FoodMenu")
                .add(R.id.companyframelayout, analytics,  tag: "Analytics")
                .hide(foodMenu)
                .hide(analytics)
                .commit()
```

Obr. 20: Vytvoření fragmentů

Dále je v kódu pojištěn případ, kdy dojde k obnovení aktivity, která má k sobě připojené fragmenty, k tomu může dojít například při rotaci zařízení nebo při přechodu na jinou kartu v mobilu. V takové situaci se pouze obnovuje reference na existující fragment, místo vytváření nového, což vede k lepšímu načítání a lepší optimalizaci.

```
// Obnovení fragmentů z FragmentManager
modelView = supportFragmentManager.findFragmentByTag( tag: "ModelView") as? Model_view
    ?: Model_view.newInstance(CompanyID)
```

Obr. 21: obnovení reference fragmentu

Dále je tu metoda setCurrentFragment() pro přepínání mezi fragmenty. Tato metoda místo aby použila metodu replace(), která ostatní skryté fragmenty zničí a pro vybraný fragment pokaždé vytváří novou instanci, tak pouze nepotřebné fragmenty skryje a chtěný ukáže a zachovává jejich instance, což opět zlepšuje optimalizaci a zlepšuje flexibilitu [12].

```
// Metoda pro měnění fragmentů
private fun setCurrentFragment(fragment: Fragment, tag: String) {
    val transaction = supportFragmentManager.beginTransaction()

    // Skryjte všechny fragmenty
    supportFragmentManager.fragments.forEach { transaction.hide(it) }

    // Pokud fragment již existuje, zobraz ho
    val existingFragment = supportFragmentManager.findFragmentByTag(tag)
    if (existingFragment != null) {
        transaction.show(existingFragment)
    } else {
        // Pokud fragment neexistuje, přidej ho
        transaction.add(R.id.companyframelayout, fragment, tag)
    }

    transaction.commit()
}
```

Obr. 22: Metoda pro přepínání fragmentů

6.5. Obsluha NavigationMenu

Navigační menu slouží jako menu, které se vysouvá ze strany a odkrývá uživatele další možnosti. Pro implementaci takového menu je zapotřebí si nastavit layout ve kterém chceme menu mít jako DrawerLayout a přidat do něj samostatný element reprezentující menu s názvem NavigationView, do kterého je však nutné přidat referenci na menu, podle kterého se definují jeho položky. V aktivitě je pak nutné získat referenci na NavigationView a podle metody setNavigationItemSelectedListener() nastavit akce po kliknutí na jednotlivé položky [13].

```
navigationView.setNavigationItemSelectedListener { item ->
    when (item.itemId) {
        R.id.manager_notifications_button -> {
            tableNotificationPopup()
            true
        }
        R.id.manager_settings_button -> {
            companySettingsPopup()
            true
        }
        R.id.manager_show_members_button -> {
            currentMembersPopup()
            true
        }
```

Obr. 23: Nastavení akcí po kliknutí na položky

6.6 Načítání dynamických dat

V aplikaci na mnoho místech probíhá získávání dat z databáze pomocí různých napsaných metod, a všechny tyto metody se musí držet určitých zásad pro správnou manipulaci. Hlavní zásadou, podle které tyto metody jsou strukturovány, je fakt, že všechny dotazy na databázi běží v kódu asynchronně, což znamená že proces, který získává data z databáze běží na samostatném nezávislém vlákně [14]. To znamená že musíme být opatrní, když chceme manipulovat se získanými daty.

Jeden ze způsobů správné manipulace s daty je použít metodu addOnSuccesListener{} a všechny manipulace dělat pouze v ní.

```
val usersRef = db.child( pathString: "companies").child(CompanyID).child( pathString: "users")
usersRef.get()
    .addOnSuccessListener { usersDataSnapshot ->
        for(user in usersDataSnapshot.children){
            user.key?.let { allUsersList.add(it) }
            val userName = user.child( path: "username").getValue(String::class.java)
            val userStatus = user.child( path: "status").getValue(String::class.java)
            if (userStatus == "online" && userName != null){
                onlineMembers.add(userName)
            }
            if (userStatus == "offline" && userName != null){
                offlineMembers.add(userName)
            }
        }
        updateAnalytics()
        loadCurrentUsersToLayout()
    }
```

Obr. 24: Ukázka využití addOnSuccesLisneru

Druhý způsob využívaný v kódu je pomocí callbacku, kde funkce zpětného volání (callback) je předána jako argument jiné funkci, která ji pak vyvolá ve vhodnou dobu během jejího provádění, aby dala vědět, že už dokončila to co měla [15]. Příklad využití je například když chceme vykreslit prvky podle načtených dat, tak musíme nejdřív počkat až se načtou všechny data a až poté spustit metodu pro vykreslení.

```
updateModel{
    drawTableOrders()
    dialog.dismiss()
}
```

Obr. 25: volání metody co využívá callback

Mezi hlavní funkcí pro nutnost správného fungování aplikace je synchronizace dat v reálném čase, aby se změny provedené jedním uživatelem ihned zobrazily všem ostatním. To je v kódu dosaženo pomocí listenerů, což je knihovní metoda od firebase, která nastaví posluchač na určitý uzel a dokáže reagovat na různé změny na tomto uzlu jako je změna nebo smazání dat [16]. Příklad použití je například u poslouchání změn u stavu stolů, a však v takovém případě je nutné zavést i metodu pro odstranění listeneru, jelikož stůl je dynamicky měnící se datová struktura a může se i smazat, a odstranění listeneru je nutné pro vyhnutí se memory leaku.

```
//Listener pro zmeny v settings
private fun listenForSettingsChanges() {
    val databaseRef = CompanyID?.let { db.child( pathString: "companies").child(it).child( pathString: "settings") }

    databaseRef?.addValueEventListener(object : ValueEventListener {
        override fun onDataChange(snapshot: DataSnapshot) {
            seatedTableNotificationPeriod = snapshot.child( path: "seatedNotification").getValue(Int::class.java) ?: 5
            eatingTableNotificationPeriod = snapshot.child( path: "eatingNotification").getValue(Int::class.java) ?: 5
            paidTableNotificationPeriod = snapshot.child( path: "paidNotification").getValue(Int::class.java) ?: 5

        }

        override fun onCancelled(error: DatabaseError) {
            Log.e( tag: "SettingsListener",  msg: "Failed to read settings", error.toException())
        }
    })
}
```

Obr. 26: Ukázka listeneru

6.7. Správa notifikací

Pro zasílaní notifikací na zařízení je nejdřív nutné vyžádat povolení o zasílání notifikací, což se dělá jednoduše v AndroidManifest.xml. V mé implementaci jsou notifikace pouze lokální, což znamená, že se posílají pouze pokud je aplikace zapnutá a běží třeba na pozadí, ale když je vypnutá, posílání není možné. Pro účely naší aplikace však lokální notifikace bohatě stačí. Počínaje systémem Android 8.0 (úroveň API 26) musí být všechna oznámení přiřazena ke kanálu. Pro každý kanál se může nastavit vizuální a sluchové chování, které se použije na všechna oznámení v daném kanálu [17]. Kanál v mém kódu je tvořen metodou createNotificationChannel(). Metoda sendNotification() následně využívá vytvořený kanál pro poslání samotné notifikace podle systémově definované builder metody [18]. V kódu je pak annonymní objekt implementující rozhraní Runnable, který ve své metodě run() vždy spustí metodu kontrolující zda je čas poslat některou notifikaci do kanálu, a pak pomocí handleru naplánuje své vlastní znovuspuštění za 5 sekund [19][20].

6.8. Editace

Editace jak u modelu podniku, tak u jídelního menu probíhá velmi podobně a používá podobné postupy, pro popis se zaměřím na editaci u modelu. Obecně se celý model ukládá v aplikaci do globální proměnné datového typu Model, která slouží jako ta hlavní do které se zapisují i nové změny při načítání modelu. Poté je tu globální proměnná editModel, do té se vždy při kliknutí na tlačítko editace zkopírují data z model pomocí metody deepCopy(). Když pak měníme model, změny se zapisují do proměnné editModel, a až po kliknutí na tlačítko save, se data z editModel překopírují do model opět pomocí metody deepCopy() a zavolá se metoda updateModel(). Toto řeření je velice pohodlné a umožňuje změny i lehce odstranit zpátky.

6.8.1. Poloha a rozměry

Scéna v modelu je reprezentována ConstraintLayoutem, tento layout vyžaduje aby všechny jeho děti (UI prvky) byli připojeny k částím layoutu nebo k ostatním prvkům pomocí takzvaných constrainů [21]. V scéně modelu jsou všechny stoly a pomocné útvary připevněny k hranám layoutu, což znamená že by měli být všechny prvky uprostřed. Ale pomocí parametrů horizontal a vertikal bias lze určovat procento šířky/výšky layoutu, tudíž lze takto libovolně měnit jeich pozici v layoutu. Toto řešení je velmi praktické, jelikož pozice prvků se přizpůsobují rozměrům a jejich pozice bude stejná na různých zaříních s různými velikostmi.

Přesouvání prvků v layout funguje pomocí metody setOnTouchListener(), která reaguje na dotikové gesto uživatele. Po kliknutí na stůl/pomocný útvar se zapne režim editace pozice, který má za úkol skrýt ostatní možnosti editace a umožnit pohyb pouze pro vybraný stůl/pomocný útvar. Když uživatel stůl poprvé dotkne, metoda si zapamatuje počáteční pozici prstu i aktuální umístění stolu v podobě relativních souřadnic. Při pohybu prstem po obrazovce metoda průběžně počítá, o kolik se uživatel posunul od počátečního dotyku. Tento posun přepočítává na relativní změnu pozice stolu vzhledem k celkové velikosti plochy. Než však novou pozici aplikuje, ověřuje, zda by stůl v nové pozici nekolidoval s jinými prvky na ploše.

Pro kontrolu kolizí vytváří pomyslný obdélník kolem navrhované nové pozice stolu a porovnává ho s obdélníky všech ostatních prvků. Pokud by došlo k překrytí, přesun se neprovede. Pouze pokud je nová pozice volná, metoda aktualizuje souřadnice stolu a překreslí jeho zobrazení. Celý tento proces zajišťuje plynulé a přirozené přesouvání stolů s okamžitou vizuální zpětnou vazbou, přičemž dbá na to, aby stoly zůstaly v hranicích plochy a vzájemně se nepřekrývaly [22].

Podobná kontrola je i když u prvku mění uživatel rozměry, kde se nejdřív spustí metoda pro kontrolu zda by prvek s novými rozměry nepřekrýval jiný prvek, nebo aby nevyčníval z layoutu, a až po vyhodnocení se aplikují nové rozměry.

6.9. Vykreslení grafů

Jak už bylo jednou zmíněno, pro vytváření grafů byla použita knihovna MPandroidChards. V každé metodě pro vykreslní takového grafu se nejprve inicializuje samotný graf (BarChart) a nastaví se mu výšku a šířka. Pro tvorbu grafu se využívájí data o první pětici s nejvyšším skoře. Každý prvek je v grafu reprezentován jedním sloupcem, kde jeho výška odpovídá počtu jeho skóre.

Vzhled grafu je upraven tak, že sloupce mají světle modrou barvu a nad každým je zobrazena konkrétní číselná hodnota počtu skóre. Pro lepší čitelnost jsou nastaveny popisky stolů na ose X pod úhlem 40 stupňů. Osa Y pak zobrazuje počty skóre s rozestupy, které se automaticky přizpůsobí maximální hodnotě v grafu [23].

7. Závěr

Cílem této aplikace bylo zjednodušit práci číšníka v restauračním provozu, což se podle mého názoru tento cíl podařilo úspěšně naplnit. Během vývoje jsem získal cenné zkušenosti s Android vývojem, zejména s prací v Android Studiu, správou dat v Firebase Realtime Database a tvorbou uživatelsky přívětivého rozhraní. Myslím si, že aplikace má konkrétní praktické využití v gastronomických zařízeních, kde může výrazně urychlit obsluhu hostů a snížit chybovost při předávání informací. Do budoucna by bylo vhodné rozšířit funkcionalitu například o integraci platebního systému, jinak jsem s výsledkem apikace spokojený.

Zdroje

[1] Android Developers. *Android Studio Overview*: https://developer.android.com/studio

[2] Kotlinlang.org. *Why Kotlin*: https://kotlinlang.org/

[3] Firebase Documentation. *Introduction to Firebase*.: https://firebase.google.com/docs

[4] MPAndroidChart GitHub. *MPAndroidChart Documentation*: https://github.com/PhilJay/MPAndroidChart

[5] Gradle Documentation: https://gradle.org/

[6] Android application file structure:

https://www.geeksforgeeks.org/android-android-apps-file-structure/?ref=next_article_top

[7] Android development guide, components: https://developer.android.com/guide/components

[8] Login and registration using firebase in android

https://youtu.be/QAKq8UBv4GI?si=nP0retBygfiiibqX

[9] Intents and intent filters

https://developer.android.com/guide/components/intents-filters

[10] Dialogs https://developer.android.com/develop/ui/views/components/dialogs

[11] Spinner https://developer.android.com/reference/android/widget/Spinner

[12] Fragments – android fundamentals:

https://youtu.be/-vAI7RSPxOA?si=ppauvL1P1vfZp3YY

[13] Navigation drawer menu:

https://youtu.be/6mgTJdy_di4?si=TNImaWeiybZuMhVT

[14] Asynchronous operations https://en.wikipedia.org/wiki/Asynchronous_operation

[15] Kotlin Callback operations made easy:

https://www.dhiwise.com/post/kotlin-callback-functions-made-easy-a-beginners-guide

[16] Retrieving data: https://firebase.google.com/docs/database/admin/retrieve-data

[17] Create and manage notification channels:

https://developer.android.com/develop/ui/views/notifications/channels

[18] Local notification in android:

https://youtu.be/LP623htmWcI?si=jBwEsVzGePZk6niI

[19] Runnable: https://developer.android.com/reference/java/lang/Runnable

[20] Handler: https://developer.android.com/reference/android/os/Handler

[21] ConstraintLayout

https://developer.android.com/reference/androidx/constraintlayout/widget/ConstraintLayout

[22] DRAG & DROP - Android Fundamentals:
https://youtu.be/8hzJNjj8Zck?si=8vdM6MgqP05hw0Bt

[23] How to create bar chart:

https://youtu.be/WdsmQ3Zyn84?si=F6HLMmWpIWVonnzh

Seznam obrázků

Obr. 1: Company menu scéna

Obr. 2: profile settings

Obr. 3: model

Obr. 4: list objednavek u stolu

Obr. 5: proces placení

Obr. 6: editace modelu

Obr. 7: editace pozice stolu

Obr. 8: editace parametrů stolu

Obr. 9: ukázka jídelního meny

Obr. 10: editace jídelního meny

Obr. 11: grafy

Obr. 12: zobrazení více dat

Obr. 13: settings ukázka 1

Obr. 14: settings ukázka 2

Obr. 15: uzel “companies”

Obr. 16: uzel “users”

Obr. 17: ukázka metody toMap()

Obr. 18: ukázka metody deepCopy()

Obr. 19: ukázka metody pro vytvoření dialogu a nastavení referencí na jeho prvky

Obr. 20: Vytvoření fragmentů

Obr. 21: obnovení reference fragmentu

Obr. 22: Metoda pro přepínání fragmentů

Obr. 23: Nastavení akcí po kliknutí na položky

Obr. 24: Ukázka využití addOnSuccesLisneru

Obr. 25: volání metody co využívá callback

Obr. 26: Ukázka listeneru